Deep Learning for Clustering: Taxonomy and New Methods

Shivangi Aneja

Supervisor:
Vladimir Golkov

Outline

1. Background
    - Types Of Clustering
    - Motivation

2. Taxonomy : Building Blocks
    - Pre-processing
    - Neural Network Branch
    - Deep Features Used for Clustering
    - Non-Clustering Loss
    - Clustering Loss
    - Combination Of Losses
    - Clustering Algorithm

3. Metrics Used

4. Proposed Taxonomy

5. Contributions and Results

6. Future Work

Outline

1. Background
   - Types Of Clustering
   - Motivation

2. Taxonomy : Building Blocks
   - Pre-processing
   - Neural Network Branch
   - Deep Features Used for Clustering
   - Non-Clustering Loss
   - Clustering Loss
   - Combination Of Losses
   - Clustering Algorithm

3. Metrics Used

4. Proposed Taxonomy

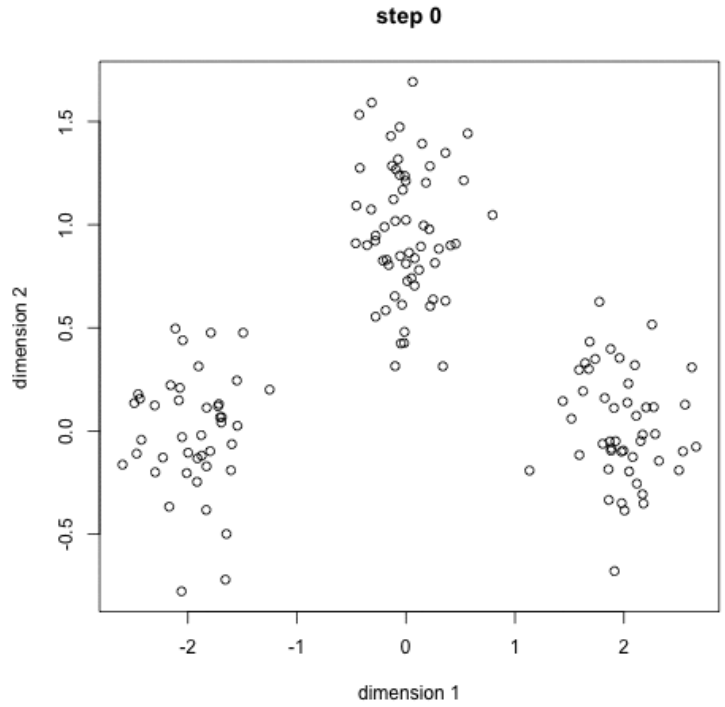
5. Contributions and Results

6. Future Work

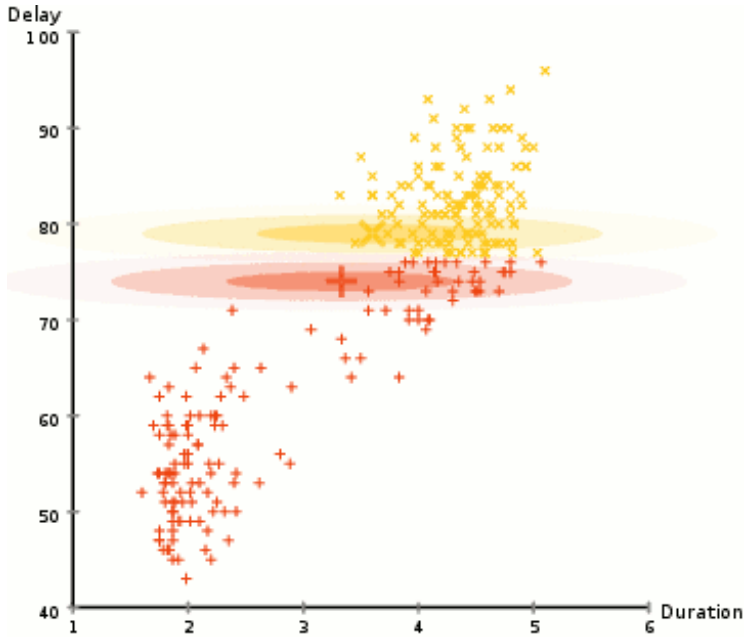
Types Of Clustering

1. Centroid Based Clustering (Fixed Number of Cluster K)

- Clusters are formed by the closeness of data points to the centroid of clusters



K-means



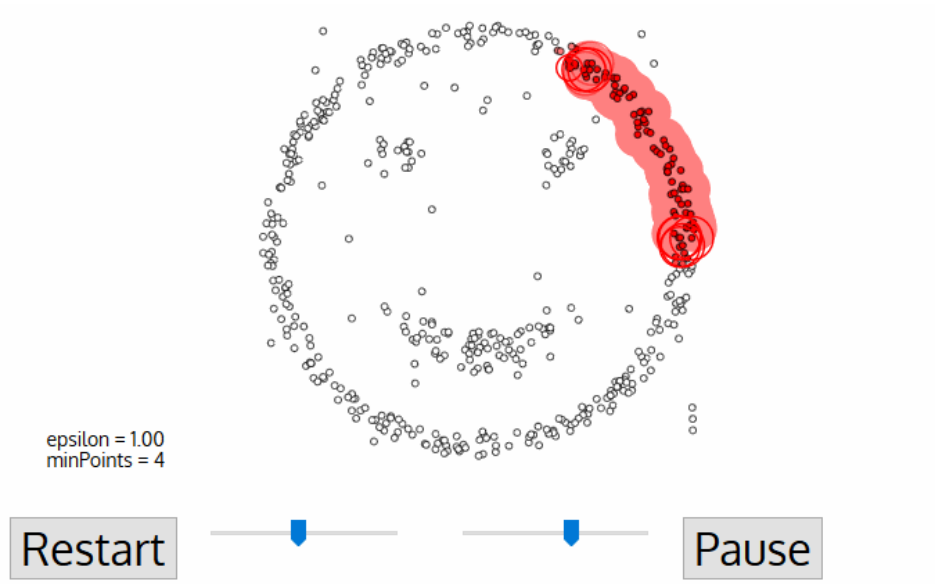
Gaussian
Mixture Models

Image Source: https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68

Types Of Clustering

2. Density Based Clustering

- Searching of data space for areas of varied density of data points in the data space . It isolates various density regions based on different densities present in the data space



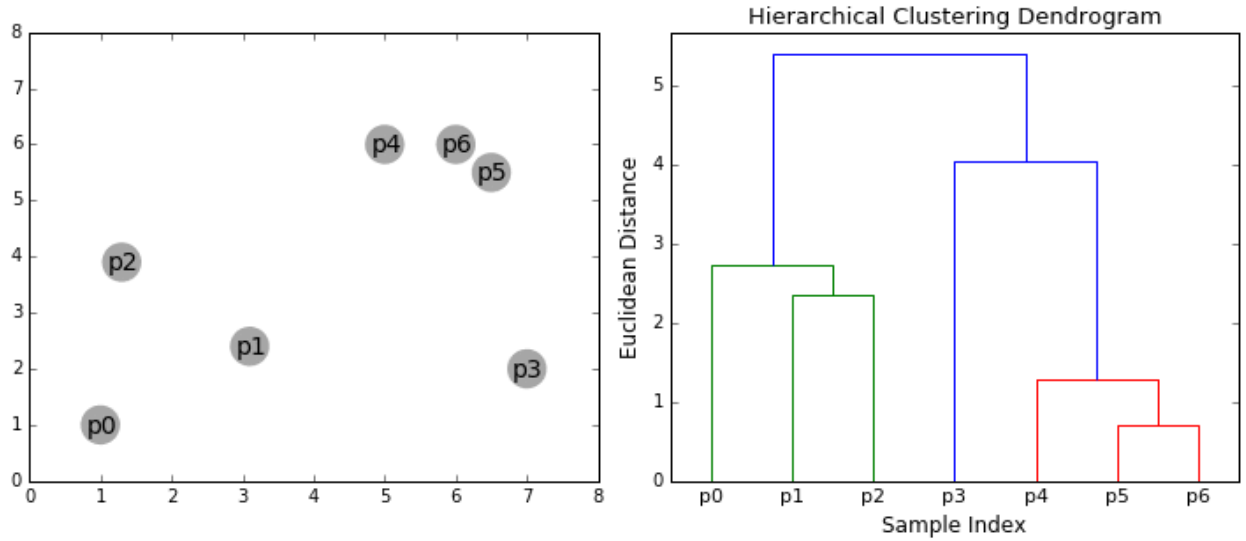
DB-Scan

Image Source: https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68

Types Of Clustering

3. Hierarchical Based Clustering

- Top-down algorithms : Each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.
- Bottom-up approach: All observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.



Agglomerative Hierarchical Clustering

Image Source: https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68

Motivation : Some Examples

1. Medical Applications
   - Can be used in identifying cancerous data set

     A Comparison of Fuzzy and Non-Fuzzy clustering Techniques in Cancer Diagnosis by X.Y. Wang and J.M. Garibaldi.

2. Search Engines
   - Search engines try to group similar objects in one cluster and the dissimilar objects far from each other. It provides result for the searched data according to the nearest similar object which are clustered around the data to be searched

     Clustering Billions of Images with Large Scale Nearest Neighbor Search by Ting Liu, Charles Rosenberg and H.A. Rowley.

3. Academics
   - Can be used to monitor the students' academic performance. Based on the students' score they are grouped into different-different clusters

     Application of k-means clustering algorithm for prediction of students' academic performance by O.J. Oyelade, O.O. Oladipupo and I.C. Obagbuwa

4. Pricing Segmentation
   - E-retailers segment the customers based on customer spent patterns and understand price sensitivity of the customers.

Outline

1. Background
   - Types Of Clustering
   - Motivation

2. Taxonomy : Building Blocks
   - Pre-processing
   - Neural Network Branch
   - Deep Features Used for Clustering
   - Non-Clustering Loss
   - Clustering Loss
   - Combination Of Losses
   - Clustering Algorithm

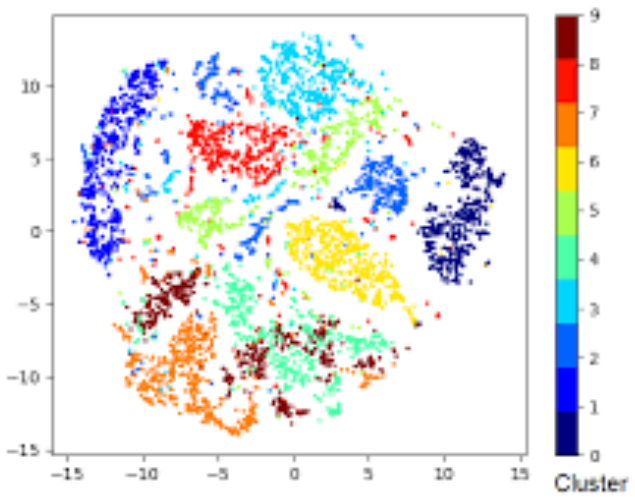
3. Metrics Used

4. Proposed Taxonomy

5. Contributions and Results

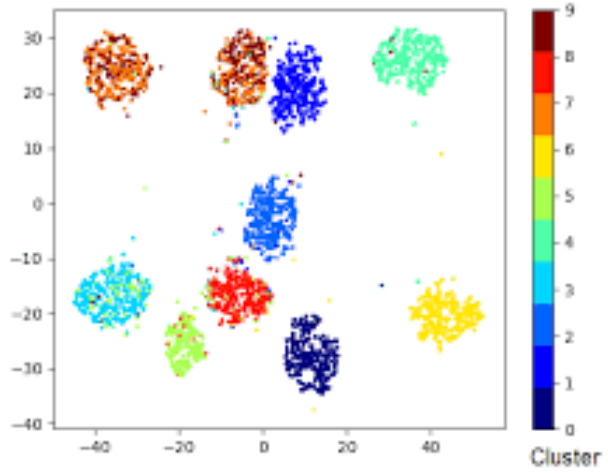
6. Future Work

Taxonomy : Basic Idea



Input Data

Deep Feature Extraction



Features

Taxonomy : Building Blocks

1. Data Pre-processing (Optional)

   This is used for feature extraction
   - Histogram Of Oriented Gradients (HOG)
   - Color Histogram

2. Architecture of Neural Network Branch

   This transforms input data
   - Multi layer perceptron
   - Convolutional Neural Network
   - Deep Belief Network
   - Generative Adversarial Network
   - Variational Autoencoder

3. Non Clustering loss

   This makes sure that data retains its original distribution
   - Autoencoder Reconstruction loss

     $$L = \sum_{i} \left|\left|x_I - f(x_i)\right|\right|^2$$
   - Self-Augmentation Loss

     $$L = -\frac{1}{N}\sum_{N} s(f(x), f(T(x)))$$

Taxonomy : Building Blocks

4. Clustering Loss

This makes sure that data learns cluster friendly representation

- K-means Clustering Loss

  Assures that data points are evenly distributed around cluster centers

- Cluster Assignment Hardening Loss

  Soft assignment of data points to clusters

- Balanced Assignment Loss

  Ensures balanced cluster assignments

5. Combination of Losses

This makes sure that data retains its original distribution as well as learns cluster friendly representation

$$L = \alpha \left(L_{Clustering}\right) + (1 - \alpha) L_{\{Non-Clustering\}}$$

- Pre-training ($\alpha = 0$) and then fine-tuning ($\alpha = 1$)
- Joint Training ($0 < \alpha < 1$)
- Variable Scheduling

Taxonomy : Building Blocks

6. Cluster Updates

This makes sure that data learns cluster friendly representation

- Jointly updated (with network parameters)
- Alternating updated with network model
  - Number of iterations
  - Frequency of updates

Outline

1. Background
   - Types Of Clustering
   - Motivation

2. Taxonomy : Building Blocks
   - Pre-processing
   - Neural Network Branch
   - Deep Features Used for Clustering
   - Non-Clustering Loss
   - Clustering Loss
   - Combination Of Losses
   - Clustering Algorithm

3. Metrics Used

4. Proposed Taxonomy

5. Contributions and Results

5. Future Work

Metrics Used

- Clustering Accuracy

$$ACC = \max_{m} \frac{\sum_{i=1}^{n} 1\{l_i = c_i\}}{n}$$

where l_i is the ground-truth label, c_i is the cluster assignment label by the algorithm

- NMI (Normalized Mutual Information)

$$H(X) = -p_i \log(p_i)$$
$$I(l, c) = H(l) - H(l|c)$$

$$NMI(l, c) = \frac{I(l, c)}{\frac{1}{2}[H(l) + H(c)]}$$

where I is the mutual information metric and H is entropy

Outline

1. Background
   - Types Of Clustering
   - Motivation

2. Taxonomy : Building Blocks
   - Neural Network Branch
   - Deep Features Used for Clustering
   - Non-Clustering Loss
   - Clustering Loss
   - Combination Of Losses
   - Clustering Algorithm

3. Metrics Used

4. Proposed Taxonomy

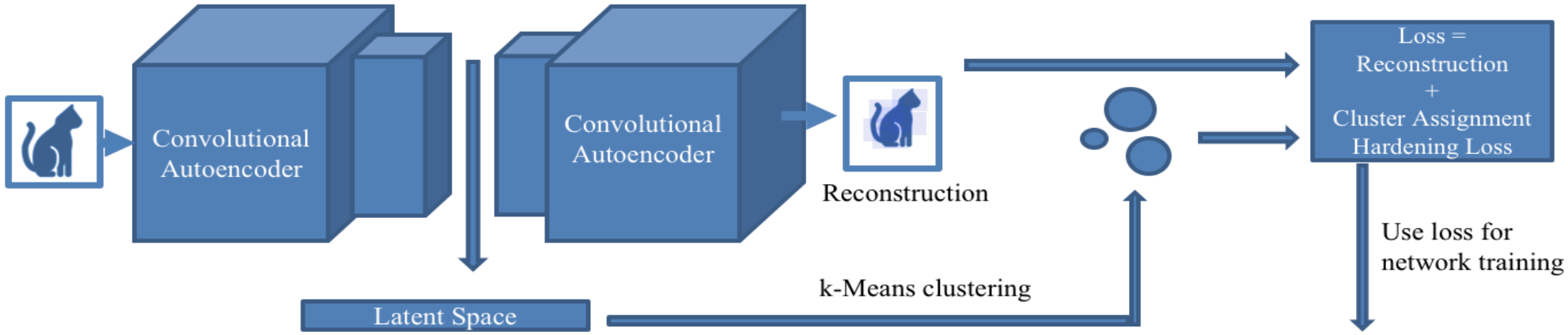
5. Contributions and Results

5. Future Work

Proposed Taxonomy

Steps :

1. Pre-train the network (CNN) with Non-Clustering loss (MSE loss).
2. Jointly train the network using Clustering loss (KL-divergence loss).
3. Evaluate results of encoder output



Outline

1. Background
   - Types Of Clustering
   - Motivation

2. Taxonomy : Building Blocks
   - Neural Network Branch
   - Deep Features Used for Clustering
   - Non-Clustering Loss
   - Clustering Loss
   - Combination Of Losses
   - Clustering Algorithm

3. Metrics Used

4. Proposed Taxonomy

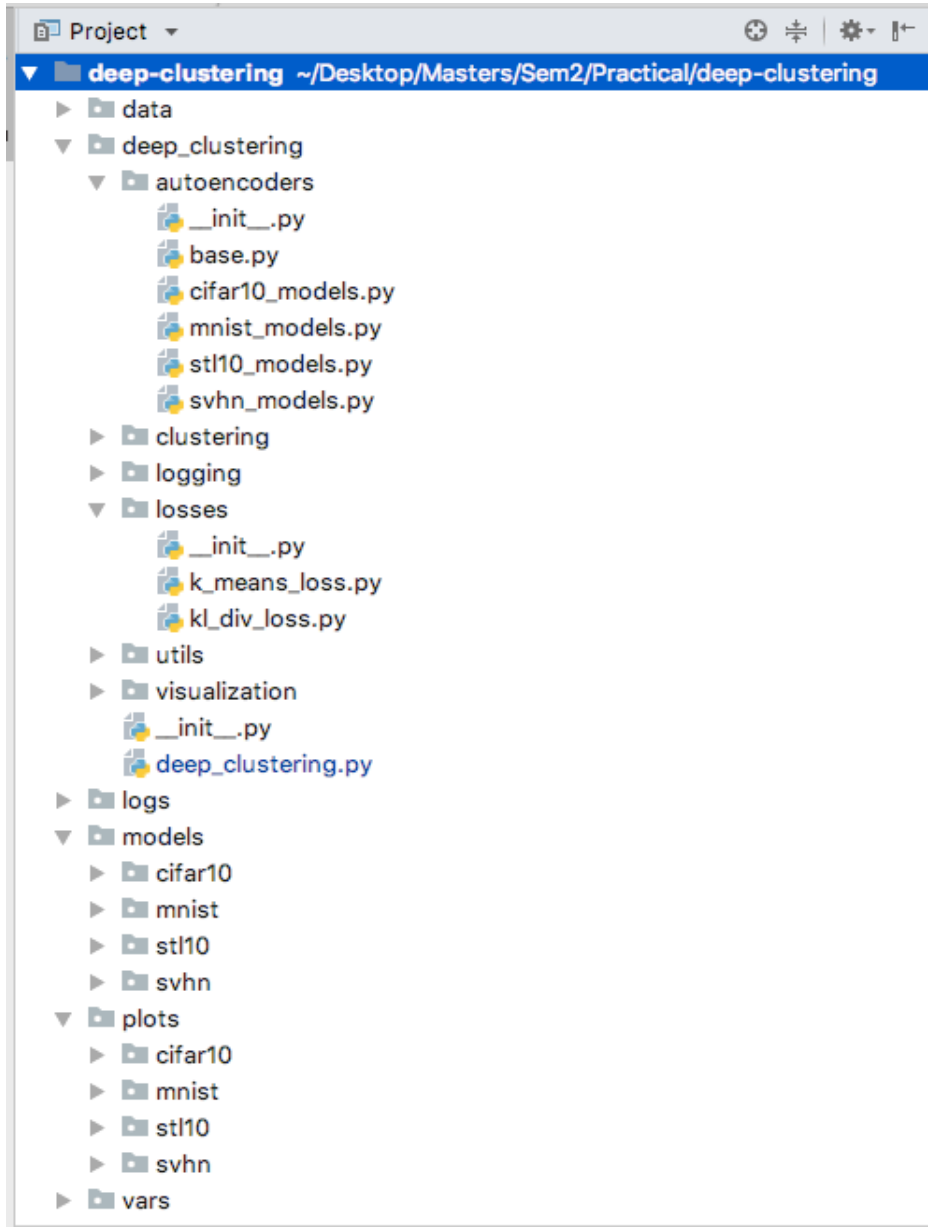
5. Contributions and Results

5. Future Work

Contributions and Results

1. Migration of code from Theano/Lasagne to Pytorch
2. Dynamic Logging (using Tensorboard)
3. Code Cleaning
   - Changed the framework to be more modular. Framework is more generic.
   - The parameters can now be passed as command-line arguments.

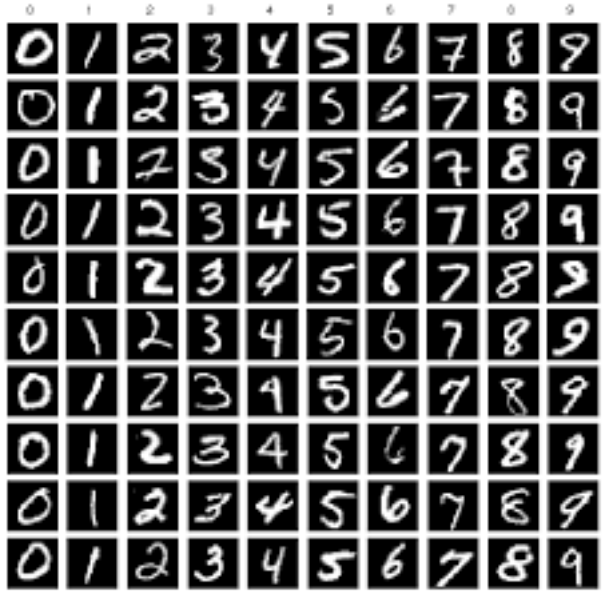
   Dataset, Autoencoder Architecture, Non-Clustering Loss, Clustering Loss, Latent Dimension, Batch Size, Epochs, Learning Rate, Regularization Strength, Dropout, Optimization Technique, Alpha, etc.
4. Tried to produce results for MNIST dataset
   - Tried out different architectures and did hyper-parameter search from scratch.
5. Evaluated taxonomy on Fashion MNIST dataset
6. Slight Improvement of results on CIFAR-10 and STL-10 dataset




```
{
'optim': 'adam',
'learning_rate': 0.0001,
'pretrain_loss': 'mse',
 'data_dirpath': 'data/',
 'latent_dim': 300, ’
'pretrain_epochs': 20,
 'finetune_epochs': 10,
'dropout': 0.2,
 'alpha': 0.5,
'autoencoder': 'fmnist_autoencoder1',
'pretrain_model_name': 'fmnist_p',
'finetune_model_name': 'fmnist_f',
'batch_size': 256,
'k_init': 30,
'finetune_loss': 'kl_div',
 'weight_decay': 0,
'dataset': 'fmnist',
 'random_seed': 1
}
```

MNIST Dataset

- 28 x 28 pixel greyscale images of hand-written digits
- Training data (55000 images)
- Validation data (5000 images)
- Test data (10000 images)



Contributions and Results : MNIST

- Architecture 5

  Conv3-32 -> Relu -> Pool2 -> Conv3-64 -> Relu -> Pool2 -> Linear -> Latent

- Architecture 7

  Conv3-32 -> Relu -> Conv3-64 -> Relu -> Pool2 -> Conv3-64 -> Relu -> Pool2 -> Linear -> Latent

- Architecture 8

  Conv3-32 -> Relu -> Conv3-64 -> Relu -> Pool2 -> Conv3-64 -> Relu -> Pool2 -> Linear -> Linear -> Latent

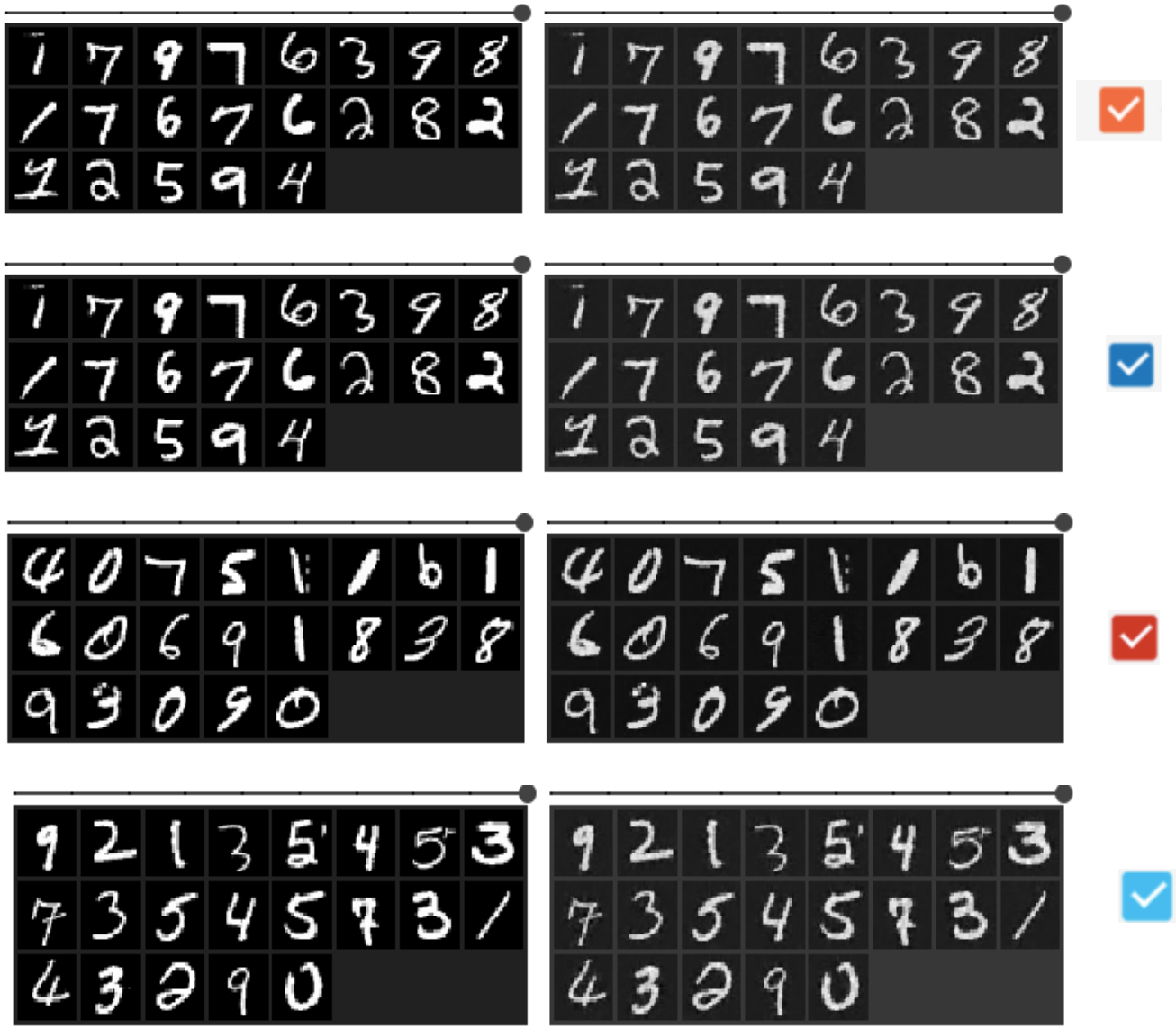
- Architecture 9 (LeNet)

  Conv5-20 -> Relu -> Pool2 -> Conv5-50 -> Relu -> Pool2 -> Linear -> Latent

- Good architectures have 2/3 Conv Layers, 2 Pool Layers, 1 Fully Connected layer

Original Image

Reconstructed Image



Architecture	Validation Accuracy	Validation NMI
5	60.4%	56.3%
7	61.9%	57.0%
8	54.5%	57.1%
9	48.7%	45.3%

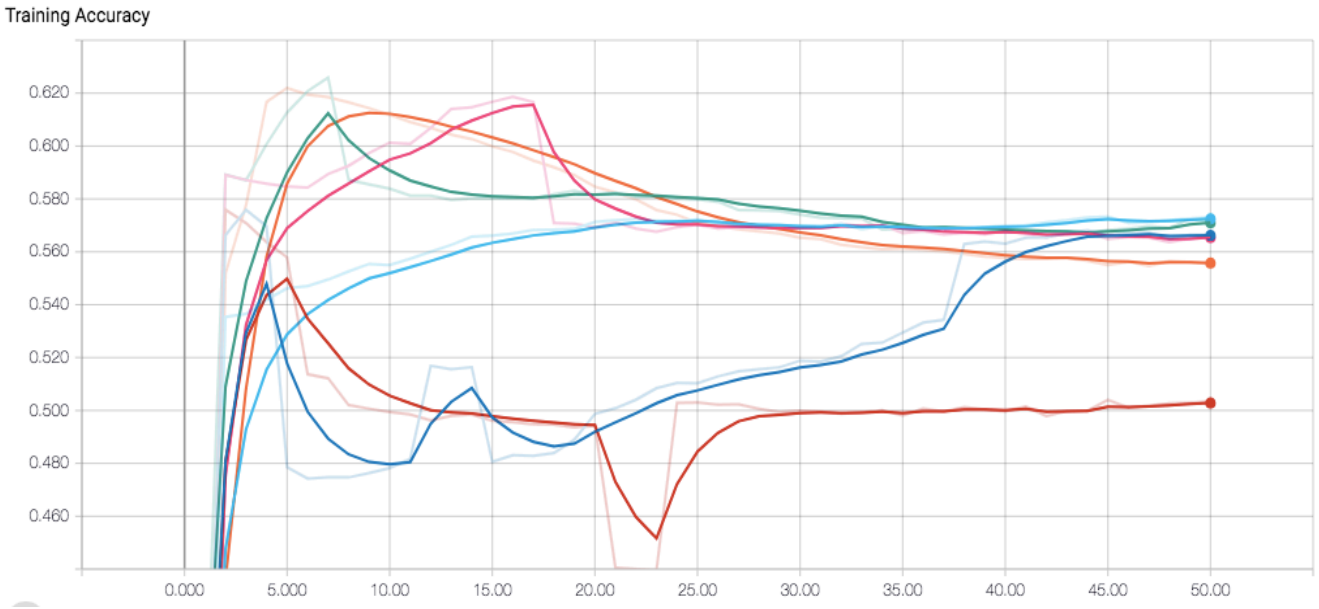
Observation 1: I decided to go with Architecture 5 and 7, and ultimately architecture 7

MNIST Architecture 7 : Hyper-parameter Search

- Architecture 7

  Conv -> Relu -> Conv -> Relu -> Pool -> Conv -> Relu -> Pool -> Linear -> Latent
- No Dropout
- No Weight Decay
- Batch Norm after Conv Layer
- Xavier Initialization of Weights
- **Tried different batch sizes**

MNIST Architecture 7 : Hyper-parameter Search



Batch Size	Validation Accuracy	Validation NMI
32	62.3%	55.3%
64	62.4%	55.1%
128	62.2%	54.4%
256	51.4%	45%
512	63.4%	54.8%
1024	56.6%	48.6%

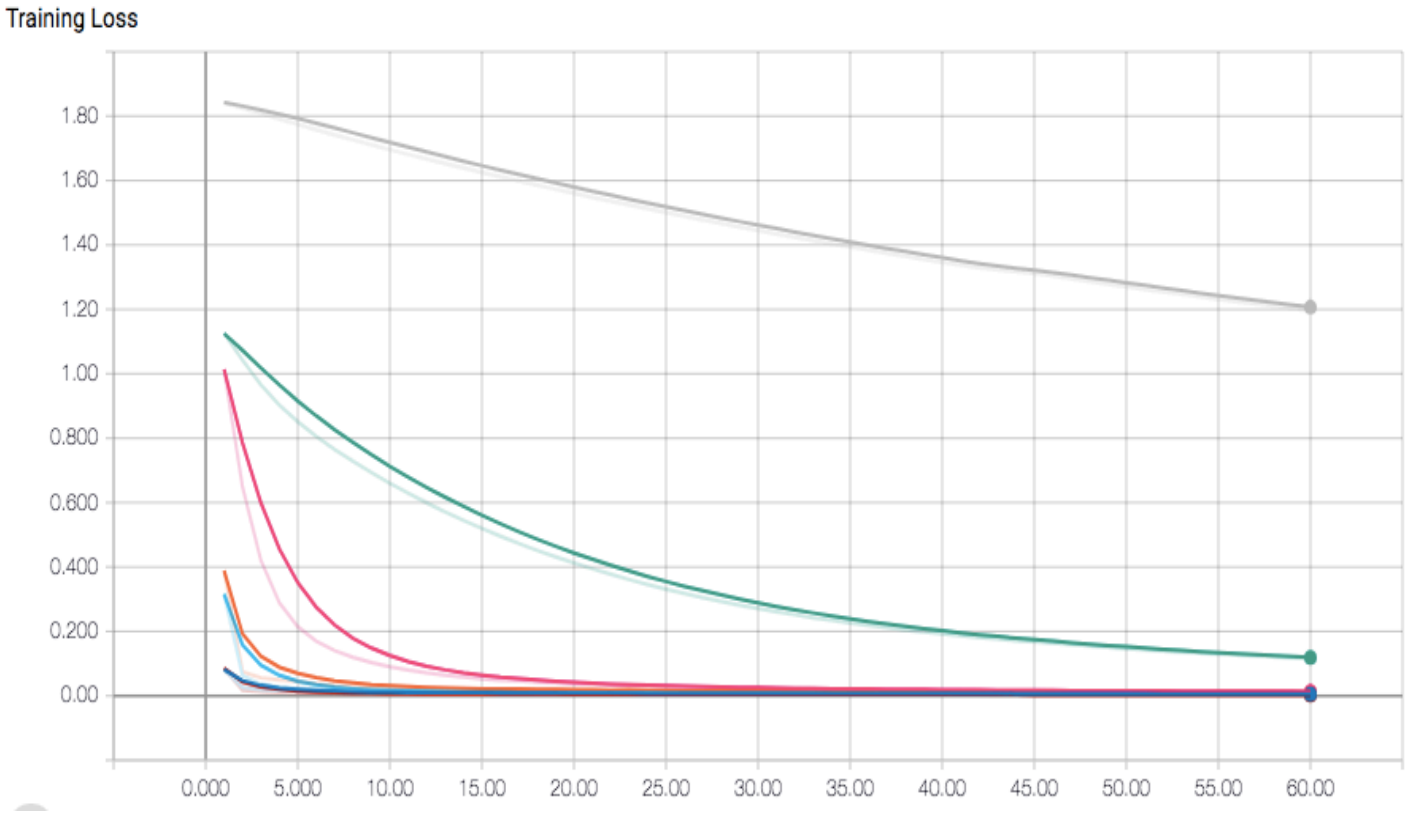
Observation 2: Not much insight from these curves but figured out later that smaller batch size works well.

MNIST Architecture 7 : Hyper-parameter Search

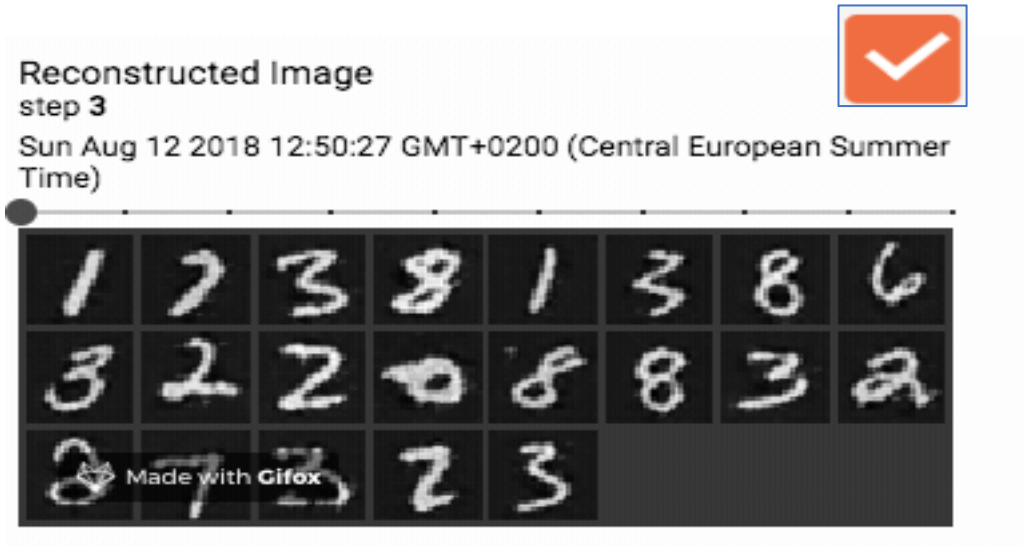
- Architecture 7

  Conv -> Relu -> Conv -> Relu -> Pool -> Conv -> Relu -> Pool -> Linear -> Latent
- No Dropout
- No Weight Decay
- Batch Norm after Conv Layer
- Xavier Initialization of Weights
- **Tried different learning rates**

MNIST Architecture 7 : Hyper-parameter Search

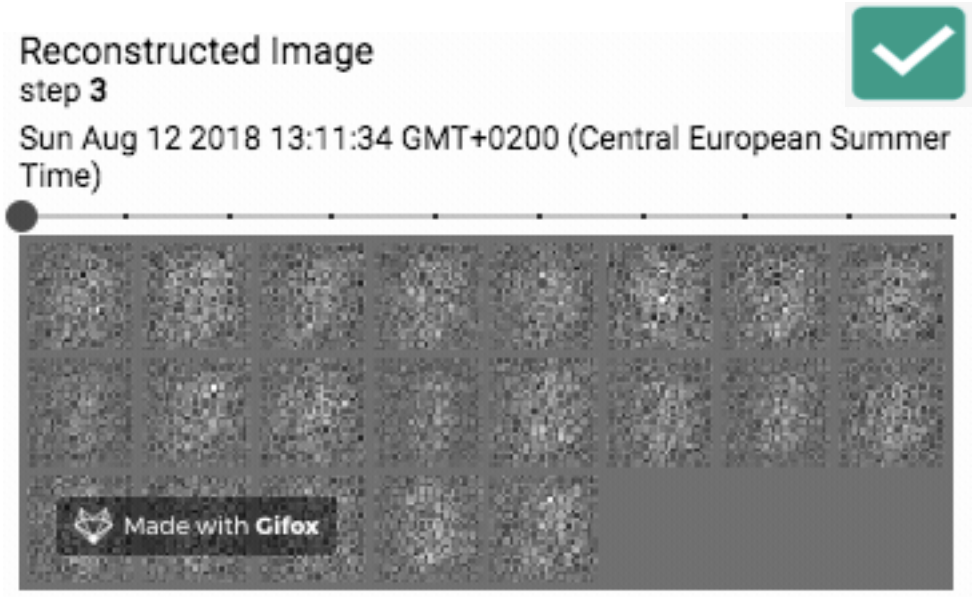


Learning Rate	Validation Accuracy	Validation NMI
1e-1	52.3%	49.9%
1e-2	55.2%	48.5%
1e-3	51.6%	50.5%
1e-4	61.0%	52.3%
1e-5	44.1%	35.3%
1e-6	60.9%	51.7%
1e-7	55.1%	49.8%

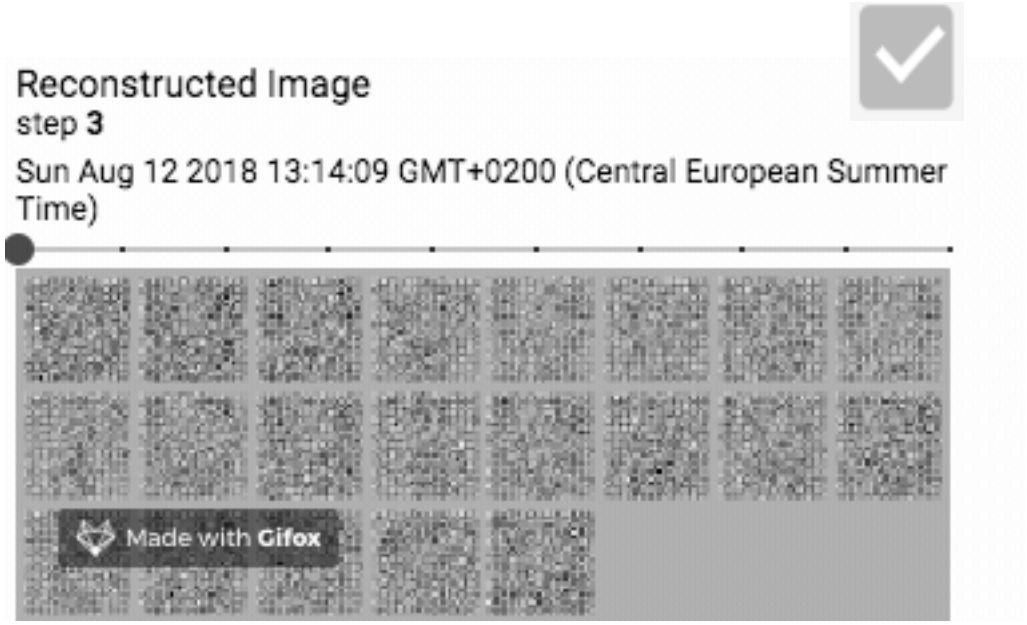


For the learning rate 1e-1

For the learning rate 1e-4



For the learning rate 1e-6



For the learning rate 1e-7

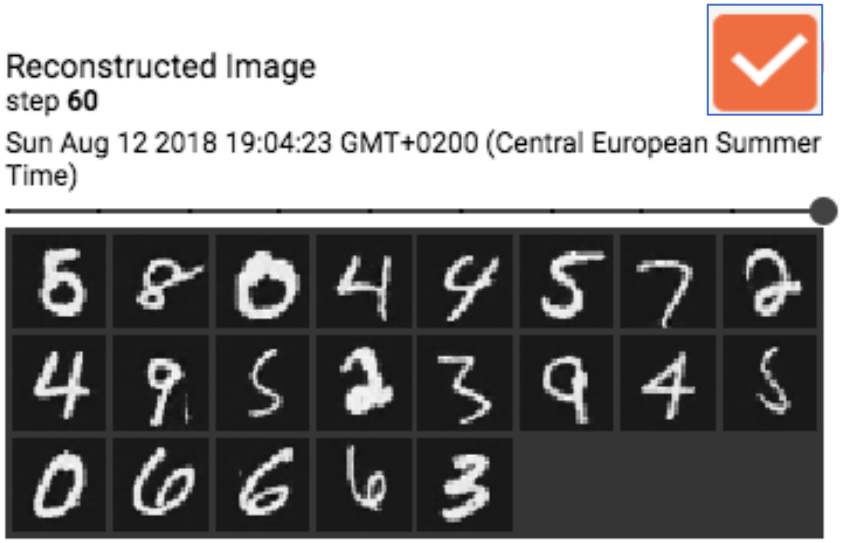
Observation 3: Very small learning rates are not good in this case

MNIST Architecture 7 : Hyper-parameter Search

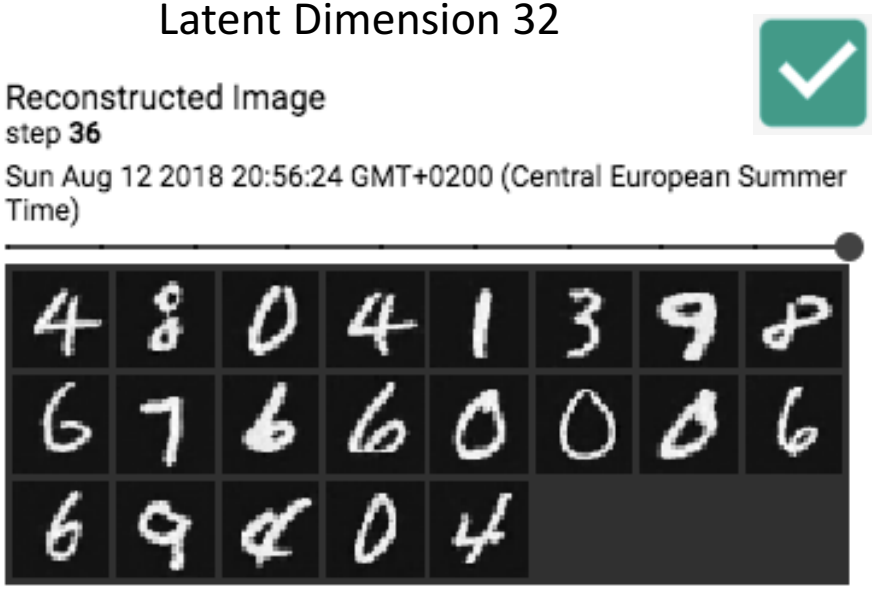
- Architecture 7

  Conv -> Relu -> Conv -> Relu -> Pool -> Conv -> Relu -> Pool -> Linear -> Latent Dim.
- No Dropout
- No Weight Decay
- Batch Norm after Conv Layer
- Learning rate of 1e-3
- Xavier Initialization of Weights
- **Tried different latent dimension**

MNIST Architecture 7 : Hyper-parameter Search



Latent Dimension 32



Latent Dimension 1024

Latent Dimension	Validation Accuracy	Validation NMI
32	57.2%	53%
64	61.5%	56.2%
128	62.5%	56.2%
256	61.7%	55.9%
512	-	-
1024	-	-

Observation 4: All of them give good reconstruction and accuracy in the same range. So I decided to go with smaller latent dimensions and train for more epochs

Contributions and Results : MNIST

The best results for pre-training that I got were with following configuration:

- Architecture 7 (3 Conv Layers, 2 Pool Layers, 1 Fully Connected layer)
- Dropout after 30 epochs
- No Weight Decay
- Batch Norm after Conv Layer
- Learning Rate of 1e-2 for first 20 epochs, then 1e-3/1e-4
- Batch Size 32
- Latent Dimension 32
- Xavier Initialization of Weights

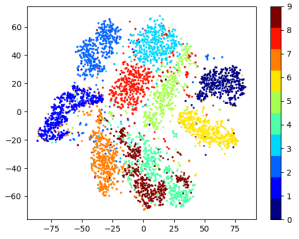
Contributions and Results : MNIST

The best results for proposed method that I got were with following configuration

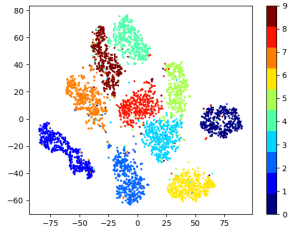
- Architecture 7 (3 Conv Layers, 2 Pool Layers, 1 Fully Connected layer)
- Used Pre-trained autoencoder from above (with latent space of 32)
- Batch Size 32
- Latent Dimension 32
- Learning Rate of 1e-3
- Use KL-divergence Loss as Clustering Loss
- Update the cluster centroids after every 5 epochs
- Use small values of alpha (< 0.5). I used 0.05

Contributions and Results : MNIST

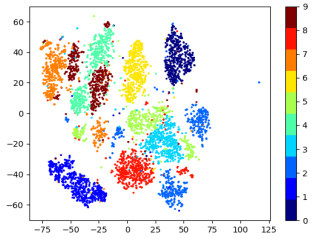
For Architecture 7



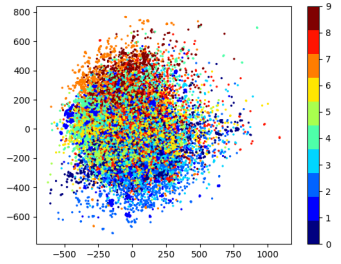
Original
Pixel Space



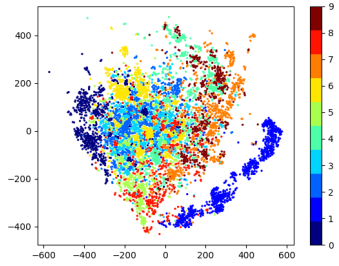
Pre-Training Results
(Latent 256)



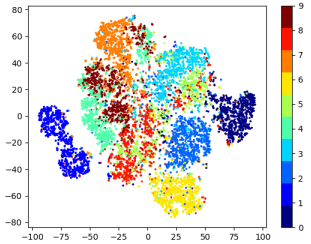
Pre-Training Results
(Latent 128)



Pre-Training Results
(Latent 64)



Pre-Training Results
(Latent 32)

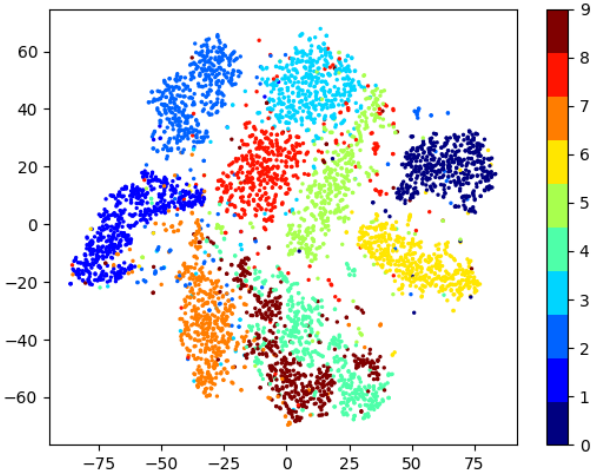


Pre-Training Results
(Latent 16)

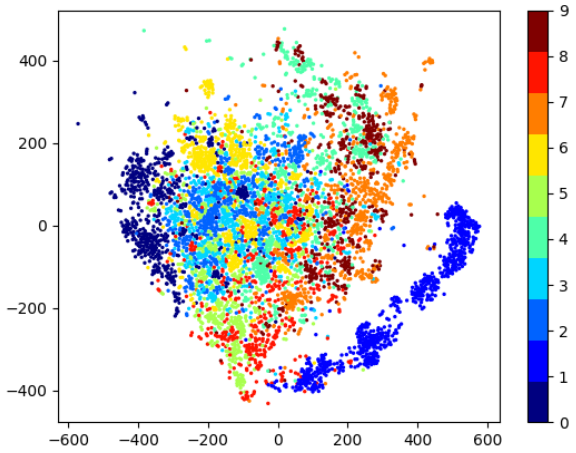
Metric	Original Pixel Space	Autoencoder (Latent 256)	Autoencoder (Latent 128)	Autoencoder (Latent 64)	Autoencoder (Latent 32)	Autoencoder (Latent 16)
Original Accuracy	54.2%	58.2%	59.8%	63.6%	**66.8%**	56.7%
Original NMI	48.0%	53.3%	54.3%	58.9%	**60%**	51.2%
T-SNE Accuracy	78.2%	79.7%	56.7%	27.8%	40.0%	63.1%
T-SNE NMI	71.6%	74.2%	51.1%	17.9%	39.5%	56.6%

Contributions and Results : MNIST

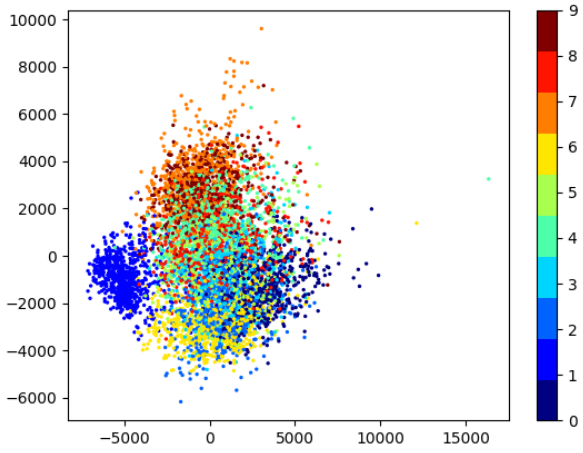
For Architecture 7



Original Pixel Space



Pre-Training Results
(Latent 32)

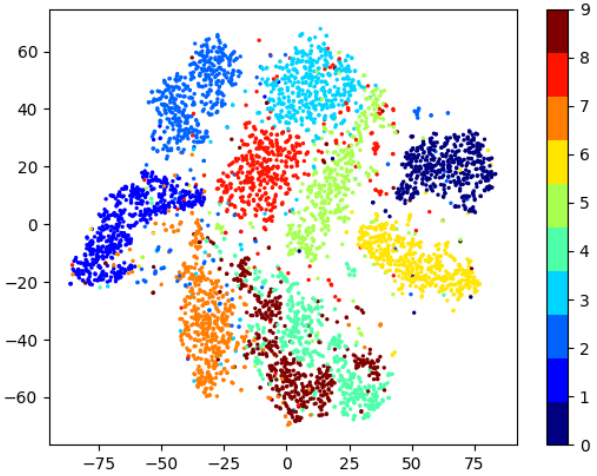


Proposed Results
(Latent 32)

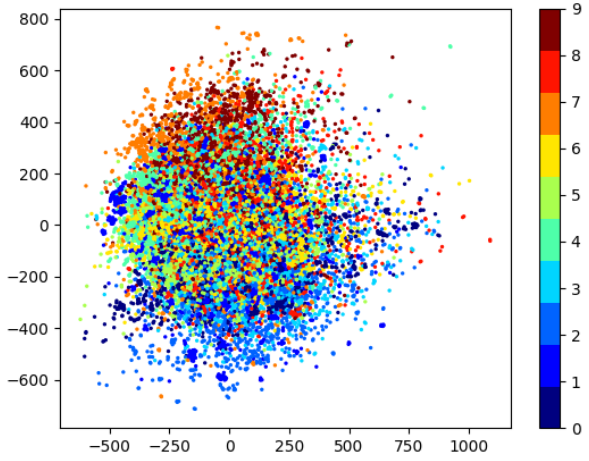
Metric	Original Pixel Space	Autoencoder (Latent 32)	Proposed (Latent 32)
Original Accuracy	51.2%	66.8%	**71.3%**
Original NMI	48.0%	60%	**63.3%**
T-SNE Accuracy	78.2%	40.0%	44.4%
T-SNE NMI	71.6%	39.5%	37.4%

Contributions and Results : MNIST

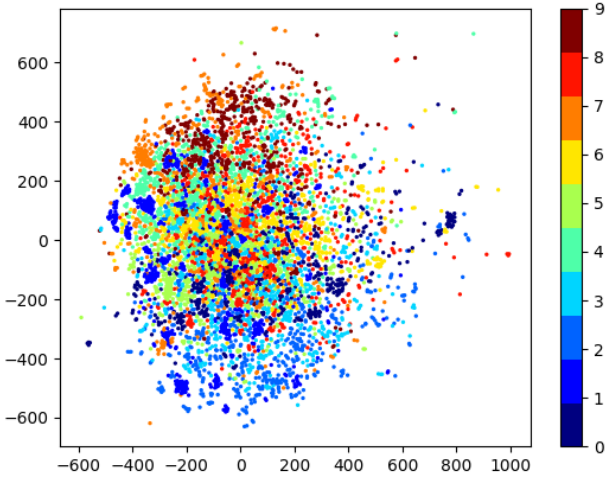
For Architecture 7



Original Pixel Space



Pre-Training Results
(Latent 64)

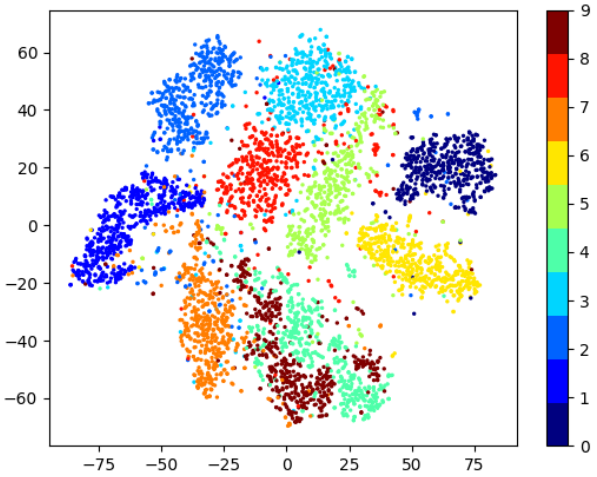


Proposed Results
(Latent 64)

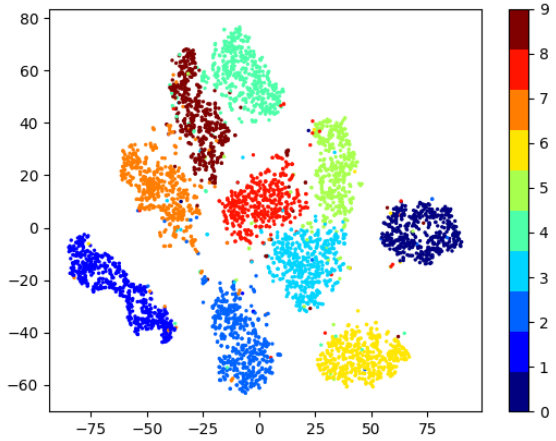
Metric	Original Pixel Space	Autoencoder (Latent 64)	Proposed (Latent 64)
Original Accuracy	54.2%	63.6%	**70.4%**
Original NMI	48.0%	58.9%	**59.0%**
T-SNE Accuracy	78.2%	27.8%	23.6%
T-SNE NMI	71.6%	17.9%	10.1%

Contributions and Results : MNIST

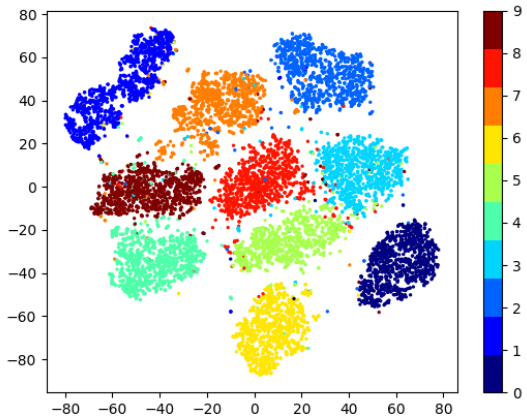
For Architecture 7



Original Pixel Space



Pre-Training Results
(Latent 256)



Proposed Results
(Latent 256)

Metric	Original Pixel Space	Autoencoder (Latent 256)	Proposed (Latent 256)
Original Accuracy	51.2%	58.2%	63.0%
Original NMI	48.0%	53.3%	56%
T-SNE Accuracy	78.2%	79.7%	**90.2%**
T-SNE NMI	71.6%	74.2%	**85.4%**

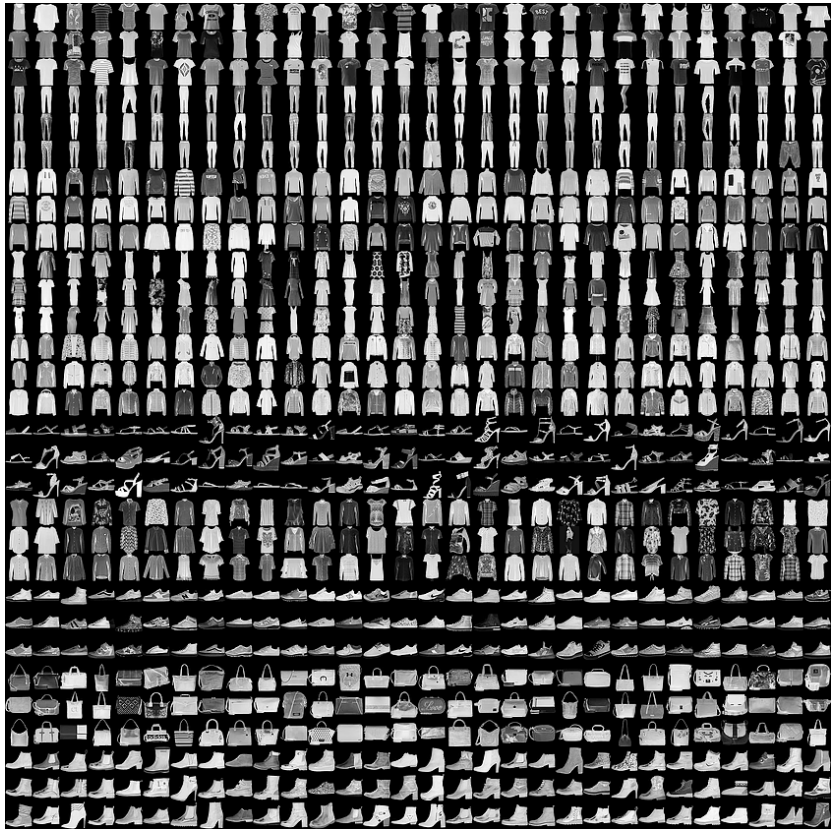
Conclusions Regarding MNIST dataset

- T-sne works well when we are projecting from very high dimensional (> 200 dimensions) to low dimensional (2/3 dimensions) space.
- Small batch size (32/64) with small latent dimension (60-100) gives good results when trained for more epochs (>50 epochs).
- A larger batch size also gives good results at times with small latent dimension.
- Learning Rate in range [1e-4, 1e-2] works best for the architectures that I have decided to go.
- The architecture that gave best results had 3 Conv layers, 2 Pool layers, and 1 fully connected layer.
- For other architectures I tried, the reconstruction looks good but the clustering accuracy/NMI are not more than 40%.
- For very small latent dimensions (16) the reconstruction is good, but the clustering accuracy is small (~50%).
- Tried K-means Clustering Loss for proposed method, but it did not improve the clustering.

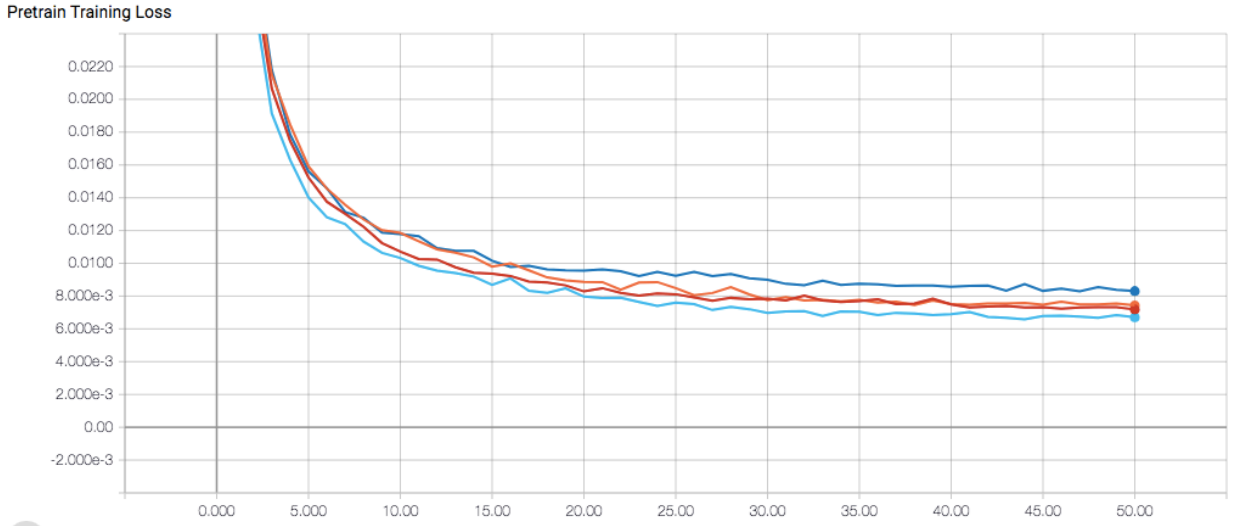
F-MNIST Dataset

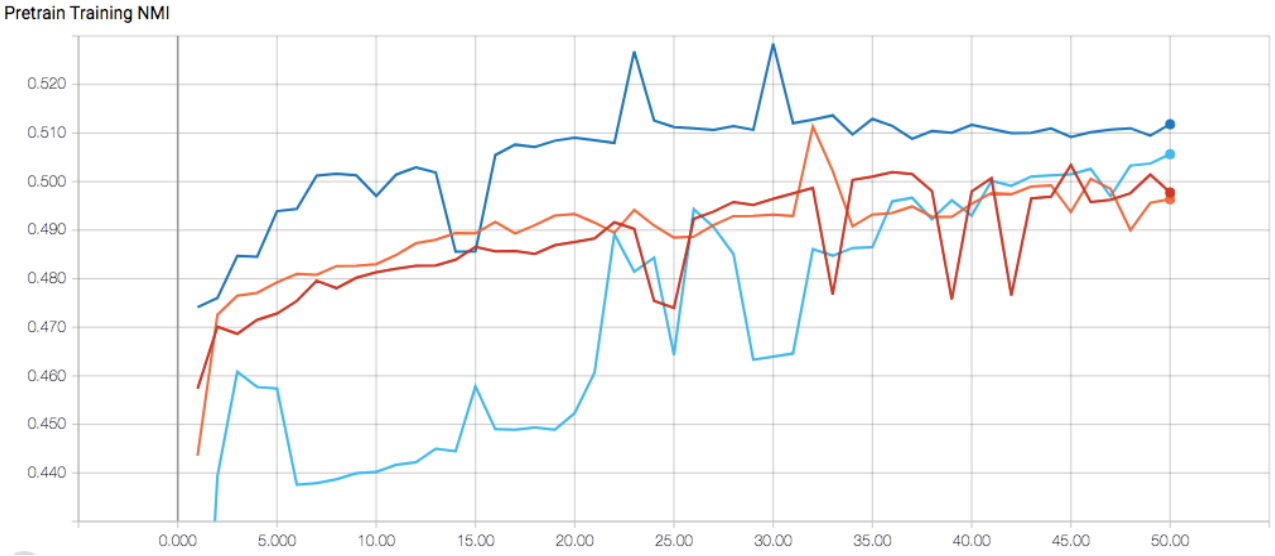
- 28 x 28 pixel greyscale images just like MNIST dataset
- Training data (55000 images)
- Validation data (5000 images)
- Test data (10000 images)
- Labels for 10 classes

  T-shirts, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags, ankle boots



Contributions and Results : Fashion MNIST

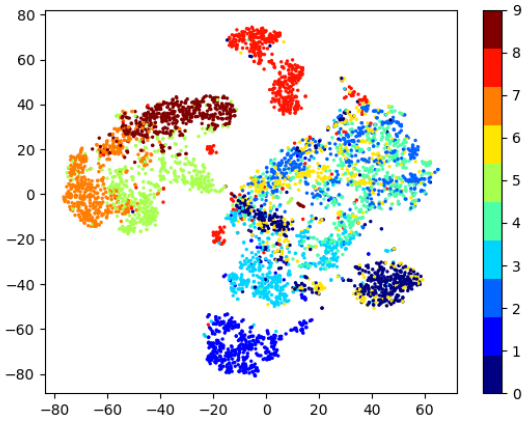




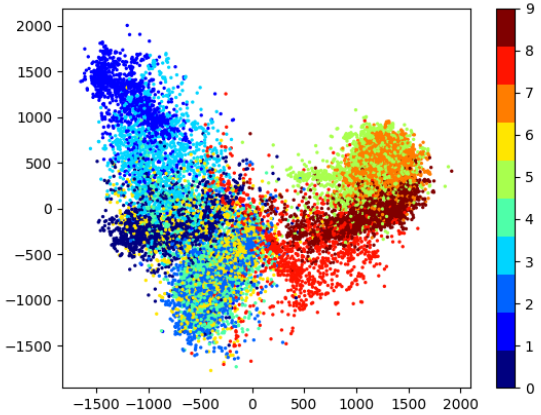
Latent Dimension	Validation Accuracy	Validation NMI
64	54.2%	56%
128	49.2%	49.2%
256	48.2%	51.0%
512	47.1%	47.9%

With the same architecture used for MNIST, tried with different latent dimensions.

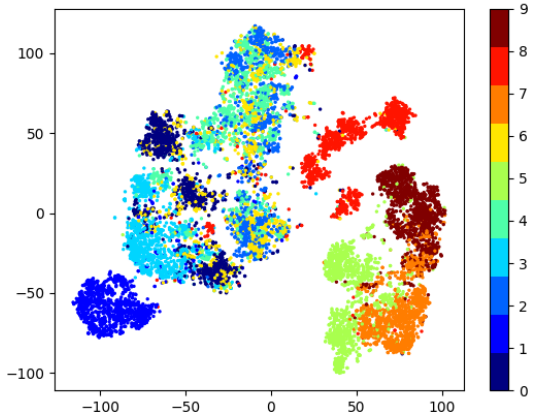
Contributions and Results : Fashion MNIST



Original Pixel Space



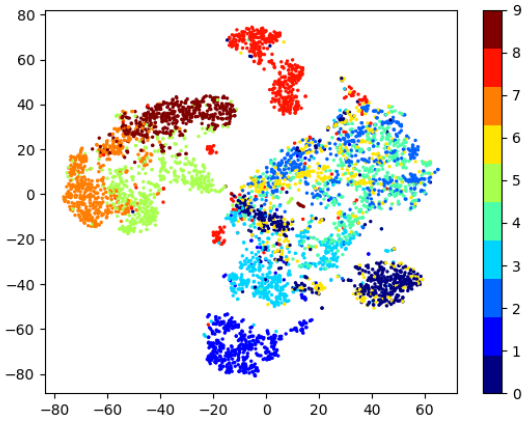
Pre-Training Results



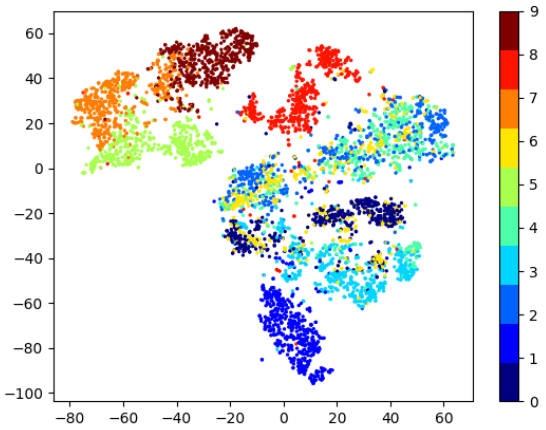
Proposed Results

Metric	Original Pixel Space	Autoencoder (Latent 32)	Proposed (Latent 32)
Original Accuracy	47%	52.6%	**57.5%**
Original NMI	50.9%	57.1%	**58.6%**
T-SNE Accuracy	57.3%	46.3%	50.7%
T-SNE NMI	55.5%	49.1%	50.8%

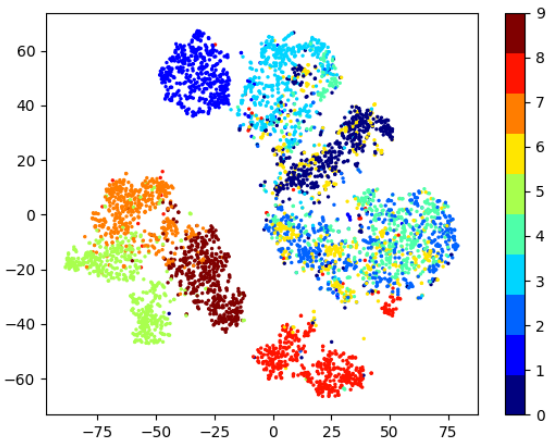
Contributions and Results : Fashion MNIST



Original Pixel Space



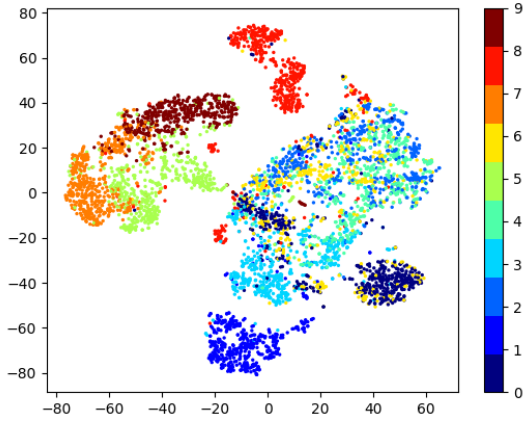
Pre-Training Results



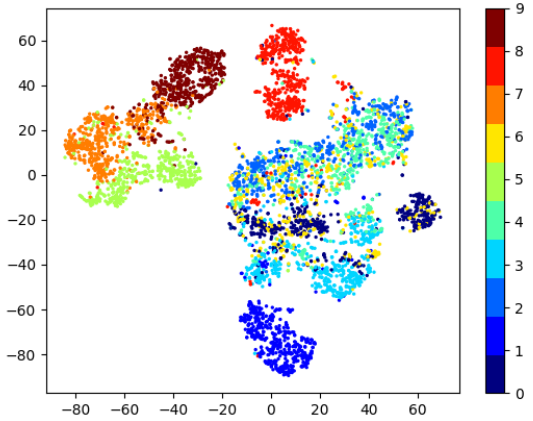
Proposed Results

Metric	Original Pixel Space	Autoencoder (Latent 64)	Proposed (Latent 64)
Original Accuracy	47%	49.1%	58.3%
Original NMI	50.9%	51.5%	54.7%
T-SNE Accuracy	57.3%	58.0%	**64.0%**
T-SNE NMI	55.5%	57.2%	**61.9%**

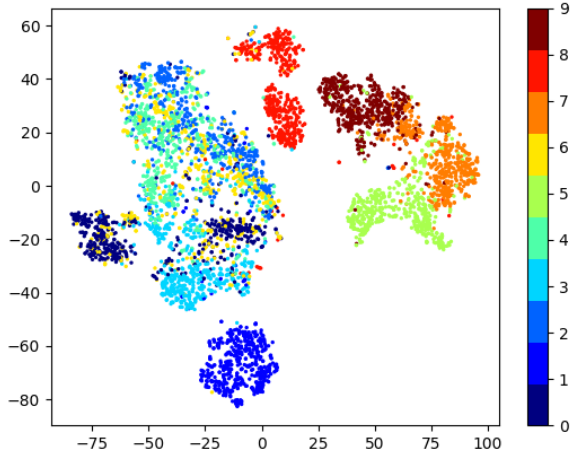
Contributions and Results : Fashion MNIST



Original Pixel Space



Pre-Training Results



Proposed Results

Metric	Original Pixel Space	Autoencoder (Latent 256)	Proposed (Latent 256)
Original Accuracy	47%	49.3%	**53.8%**
Original NMI	50.9%	50.6%	**51.6%**
T-SNE Accuracy	57.3%	42.3%	52.1%
T-SNE NMI	55.5%	47.1%	53.5%

Comparison with DEC (Deep Embedded Clustering)

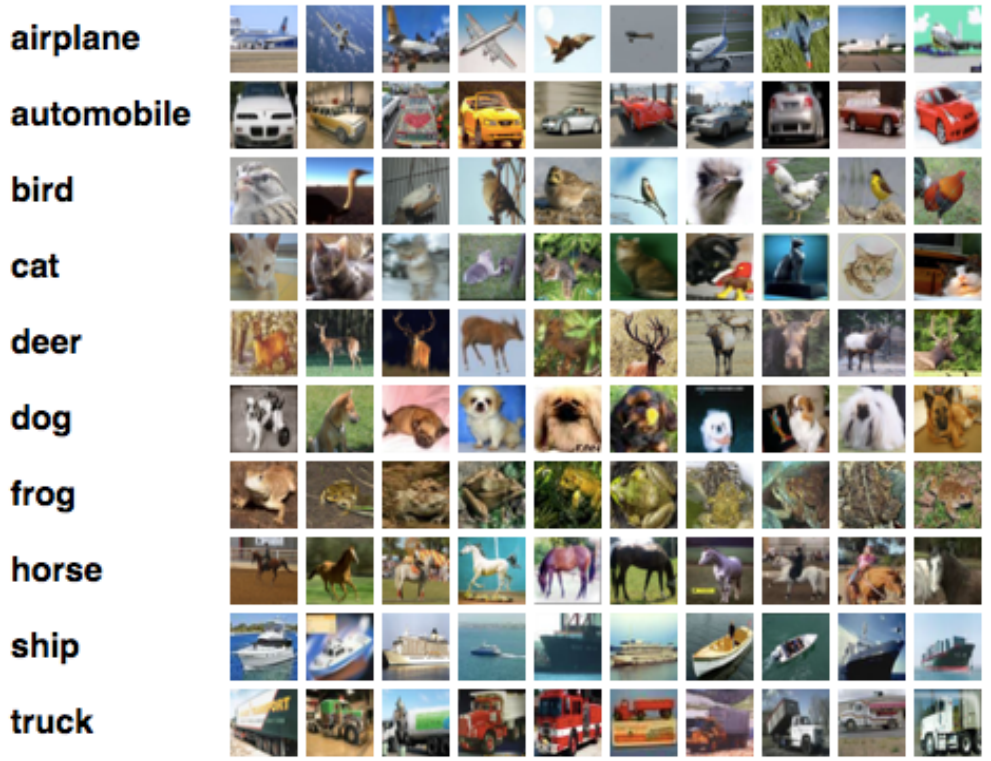
Metric	DEC MNIST	Proposed MNIST	DEC F-MNIST	Proposed F-MNIST	Δ DEC	Δ Proposed
Accuracy	84.3%	71.3%	62%	57.5%	22.3%	13.8%
NMI	80.0%	63.3%	65%	58.6%	15%	4.7%

These are the results from 32-dimensional latent vector.

CIFAR-10 Dataset

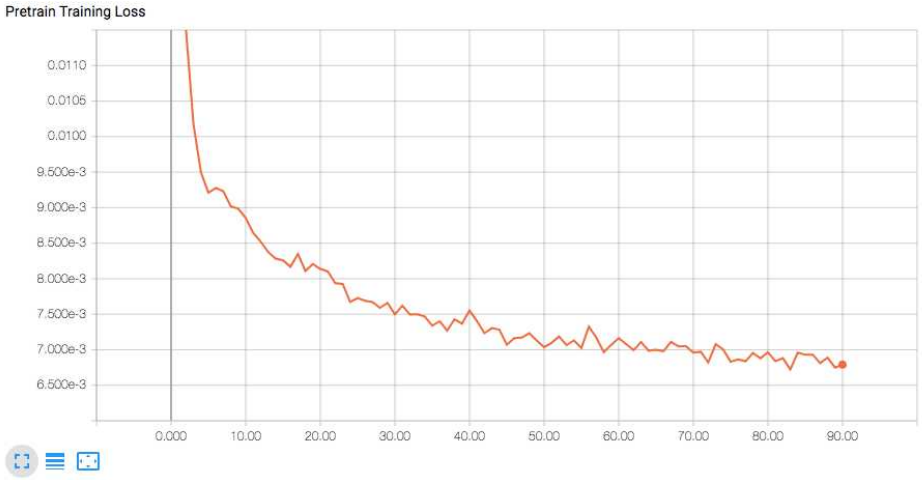
- 32 x 32 pixel RGB images
- Training data (55000 images)
- Validation data (5000 images)
- Test data (10000 images)
- Labels for 10 classes

  airplane, automobile, bird, cat, deer,
  dog, frog, horse, ship, truck



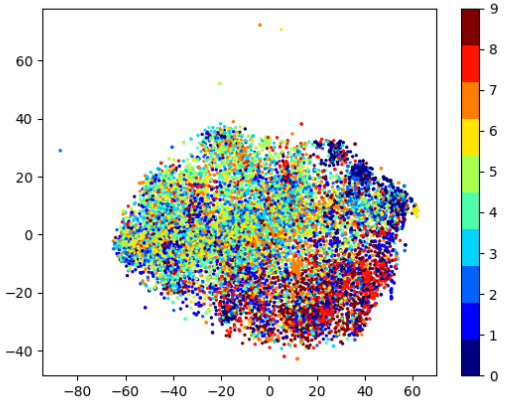
Contributions and Results : CIFAR-10

On Original Dataset



Original Image

Reconstructed Image

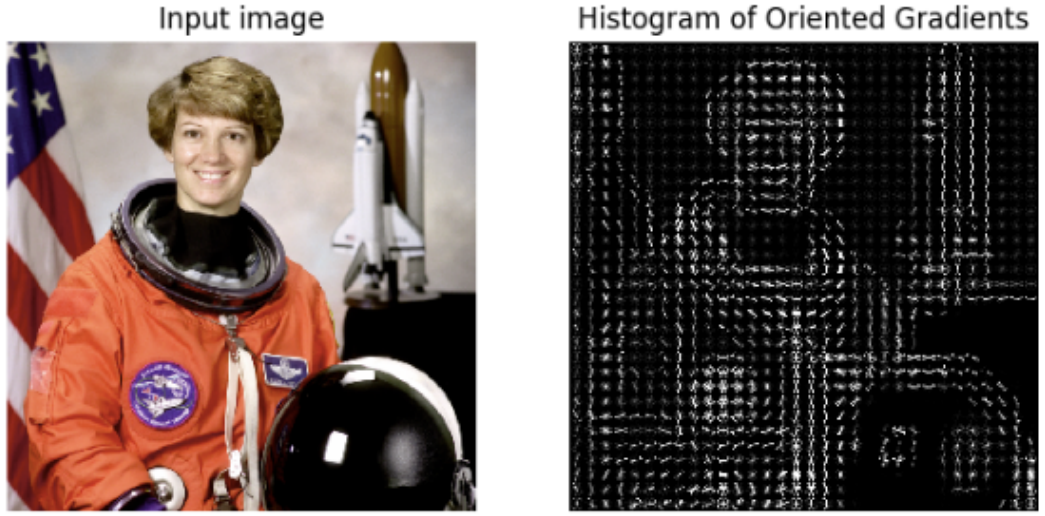


Pre training Results

Even though the loss goes down and reconstruction looks good, but the Clustering accuracy and NMI are always < 20%

Histogram Of Oriented Gradients (HOG)

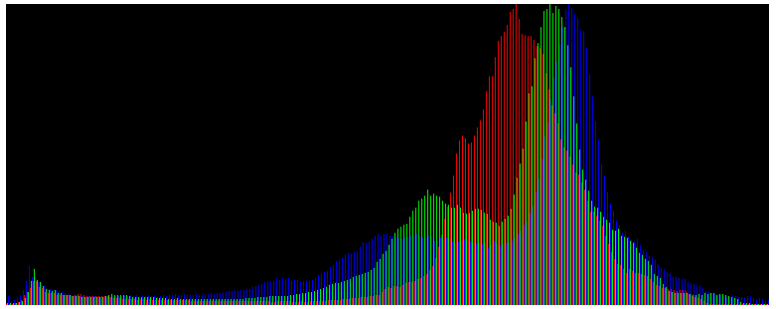
- Feature extraction method
- Captures shape of the structure in the region by capturing information about gradients



Color Histogram using HSV (Hue, Saturation, Value)

- Represents color distribution in the image

Image



Color Histogram

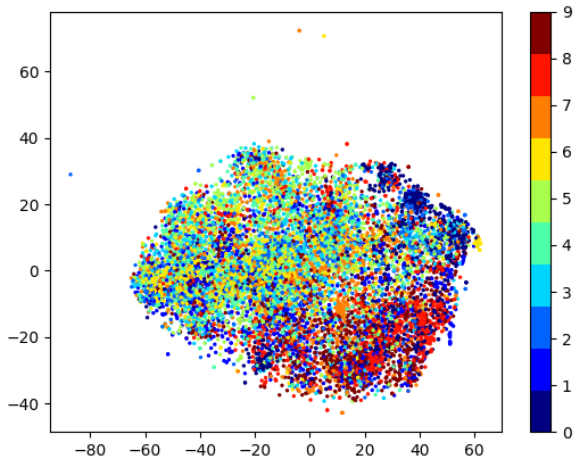
Frequency

Contributions and Results : CIFAR-10

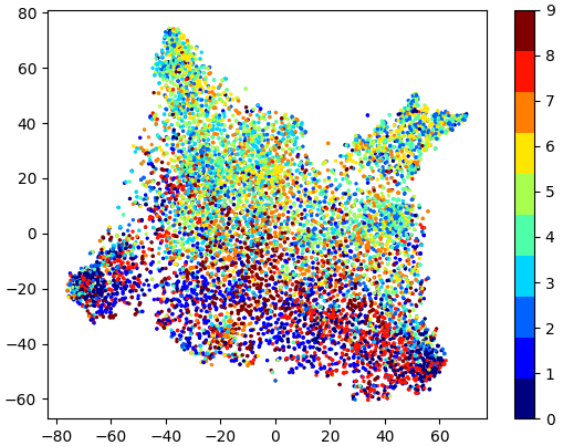
- Architecture 5
- No dropout
- Weight Decay of 1e-4
- Learning Rate of 1e-3 for first 10 epochs, then 1e-4/1e-5
- Batch Size 50
- Latent Dimension 32
- Xavier Initialization of Weights
- K-means : 30 times
- KL-divergence loss as Clustering loss
- Small values of alpha

Contributions and Results : CIFAR-10

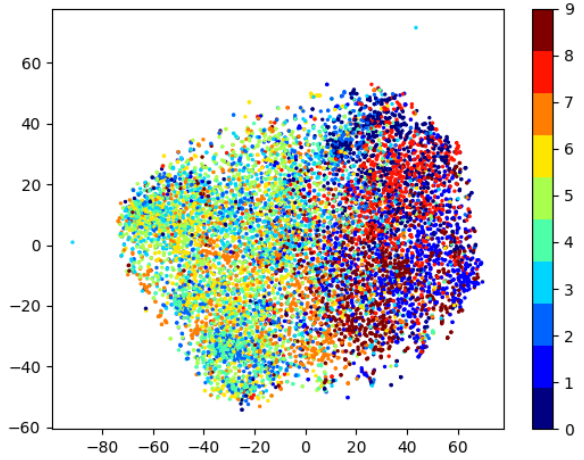
Use HOG and Color Histogram as Feature Vector to find deep representation



Original Pixel Space



Pre-Training Results



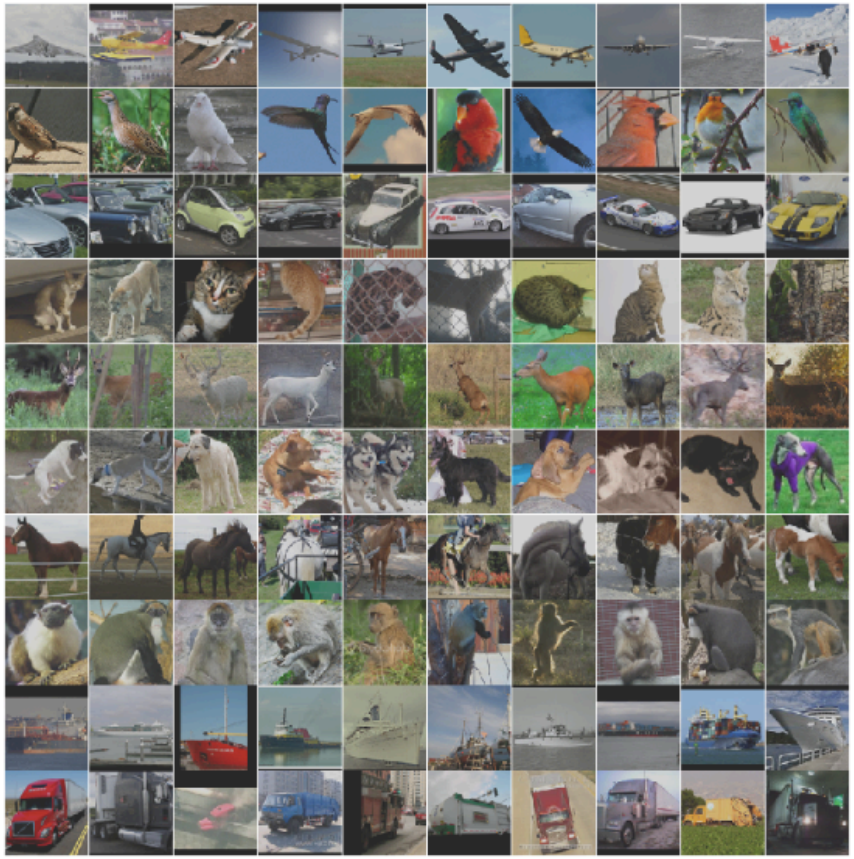
Fine-Tuning Results

	Original Pixel Space	Autoencoder	Proposed	Last Year Results
Original Accuracy	20.6%	23.6%	**26.7%**	19.5%
Original NMI	7.9%	12.4%	**18.9%**	6.6%

STL-10 Dataset

- Inspired by CIFAR-10 dataset
- 96 x 96 pixel RGB images
- Training data (500 images per class)
- Test data (800 images per class)
- Unlabeled data (100,000 images)
- Labels for 10 classes

  airplane, bird, car, cat, deer, dog,
  horse, monkey, ship, truck

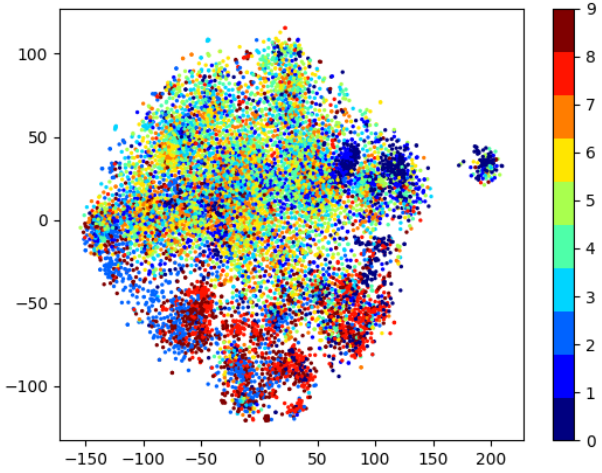


Contributions and Results : STL-10

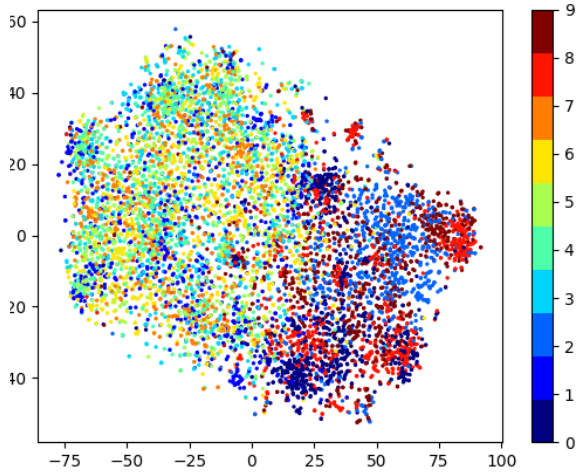
- Architecture 4
- Trained it with unsupervised data for few epochs
- No dropout
- Weight Decay of 1e-4
- Learning Rate of 1e-3 for first 10 epochs, then 1e-4/1e-5
- Batch Size 32
- Latent Dimension 100
- Xavier Initialization of Weights
- K-means : 30 times
- KL-divergence loss as Clustering loss
- Small values of alpha

Contributions and Results : STL-10

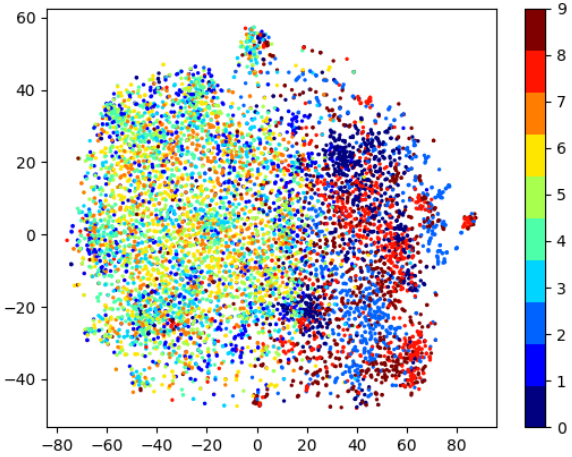
Use HOG and Color Histogram as Feature Vector to find deep representation



Original Pixel Space



Pre-Training Results



Fine-Tuning Results

	Original Pixel Space	Autoencoder	Proposed	Last Year Results
Original Accuracy	23.1%	26.3%	**28.3%**	23.0%
Original NMI	8.3%	21.7%	**22.2%**	15.4%

Conclusions

- Taxonomy works good on greyscale images.
- For RGB images, pre-processing is required.
- Improvisation for RGB images is required as its nowhere near state-of-the-art.

Future Work

- Improve accuracies further especially on RGB images.
- Look for other pre-processing techniques.
- Maybe : Transfer Learning for Feature Extraction (use Inception_v3 , Resnet) for RGB image dataset (STL-10, CIFAR-10).
- Evaluate the taxonomy on bigger datasets (e.g. Image-net).
- Evaluate the taxonomy on textual datasets (e.g. Reuters).

THANK YOU